UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

19|20

Trabajo fin de grado

Control de Sistemas Dinámicos Mediante Procesos Gaussianos

Pablo Gil Castrillo

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Control de Sistemas Dinámicos Mediante Procesos Gaussianos

Autor: Pablo Gil Castrillo
Tutor: Daniel Hernández Lobato

junio 2020

Pablo Gil Castrillo
Control de Sistemas Dinámicos Mediante Procesos Gaussianos

Pablo Gil Castrillo
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

Agradecimientos

En primer lugar, quiero agradecer a mi tutor la oportunidad de hacer este trabajo, así como la atención que me ha prestado durante todo el proceso. También agradecer a mi familia su apoyo constante durante toda la carrera y más en específico, durante el desarrollo de este trabajo. Quiero hacer una mención especial a mis amigos de siempre, que me han ayudado a desconectar y de los que estoy inmensamente agradecido por acompañarme durante tanto tiempo. Por último, agradecer a Alfredo, Bravo, Dani, David y Santi por la cercanía y el apoyo durante todo el grado.

Resumen

Los problemas de control representan una categoría en el ámbito del aprendizaje automático especializada en el manejo de un sistema dinámico mediante distintas técnicas. La categoría más comúnmente utilizada para resolver este tipo de problemas es el aprendizaje por refuerzo, en el que se ajusta el aprendizaje en base a la calidad de las acciones que realiza el controlador, medida con una función de coste. Desde el planteamiento de este tipo de problemas, ha habido distintas aproximaciones para abordarlos, pero en la mayoría de casos se repetían los mismos problemas, mayormente derivados del número de intentos necesarios durante las simulaciones para que el controlador aprendiese, lo que significa una alta repercusión en el entorno y un tiempo elevado, imposibilitando su utilización en robots u otros aparatos de bajo coste o con una vida útil corta.

Durante este TFG se desarrolla una idea planteada por Carld Edward Rasmussen y Marc Peter Deisenroth llamada PILCO. En ella se aborda la creación de un sistema que aprende a resolver problemas de control utilizando técnicas de aprendizaje automático poco comunes en este tipo de problemas, como los métodos basados en modelos.

La mayor virtud de PILCO reside en la eficiencia de datos y la velocidad en la que el controlador aprende su tarea asignada. Esto es posible gracias al funcionamiento del sistema propuesto, por el cual se obtiene información del entorno con la que se crea una imitación del mismo en los aspectos que influyen en las dinámicas que sigue el objeto a controlar. Mediante este sistema, se puede simular de forma artificial el entorno, permitiéndonos obtener datos acerca del efecto de las acciones realizadas por el controlador con las que entrenarlo, reduciendo el número de datos necesarios del entorno y aumentando la velocidad de aprendizaje del controlador radicalmente.

Este sistema utiliza Procesos Gaussianos para modelar el entorno, aprendiendo las dinámicas del sistema y sirviendo de guía al controlador durante su aprendizaje. También han sido utilizadas redes neuronales para representar el rol de controlador. Además, cabe destacar la utilización del framework TensorFlow durante toda la implementación del trabajo, ampliando la base recibida durante el grado con conocimientos directamente aplicables en otros tipos de problemas de aprendizaje automático.

Palabras clave

Aprendizaje automático, Problema de control, Controlador, Método basado en modelos, Procesos Gaussianos, Eficiencia de datos

Abstract

Control problems represent a category in machine learging specialized in the control of a dynamic system aplying diverse techniques. Most common strategy used to solve this type of problems is reinforcement learning, in wich learning is adjusted based on the quality of the actions that the controller makes, measured by a cost function. Since these problems arose, there have been different approaches to solve them, but in most of them the same problems rised again and again: the high quantity of necesary attemps during the simulations to make the controller learn his purpose as well as the time this action takes, wich disables using it on robot or other low cost or short term live applications.

During this project, the idea developed by Carld Edward Rasmussen and Marc Peter Deisenroth by the name PILCO is applied. It addresses the creation of a system that learns to solve control problems by using uncommon machine learnig tecnics such as model based methods.

PILCO's highest virtue is the data-efficiency and speed in wich the controller learns its task. This is possible thanks to the operation of the proposed system, by wich the enviroment information is used to create an immitation of it in the aspects that influence the dynamics of the object to be controlled. Through this system its possible to simulate the enviroment artificially, allowing us to obtain data about the effect of actions determined by the controller and use them to train it, reducing the number of data needed from the real enviroment and increasing controller learning speed radically.

This system uses Gaussian Processes to model the enviroment, learning the system dynamics and guiding the controller during his learning. Also, there have been used neural networks to play the controller rol. In addition, it's worth noting the use of TensorFlow framework throughout the implementation of the project, expaning the knowledge base received during the degree with new knowledge directly applicable to other types of machine learnign problems.

Keywords

Machine learning, Control problem, Controller, Model base method, Gaussian process, data-efficiency

Índice

Listas

Lista de algoritmos

Lista de códigos

Lista de ecuaciones

Lista de figuras

1

Introducción

Los problemas de control conforman una categoría en el ámbito del Aprendizaje Automático que combina técnicas de control inteligente y teoría de control en su solución. Para ello se suele utilizar el aprendizaje por refuerzo, en el que se realizan simulaciones, donde se tiene una indicación de cómo de bien está actuando el objeto de la simulación utilizada para ajustar el aprendizaje (para más información consultar [2] [3]).

El aprendizaje por refuerzo conforma un conjunto de técnicas utilizadas a fin de que un sistema aprenda en un entorno interactivo. El aprendizaje consiste en una prueba de ensayo-error, en la que el sistema, en lugar de fijarse en la coincidencia entre el resultado obtenido tras la introducción de un input (como ocurre en los problemas de clasificación), se fija en una función de coste que indica la calidad de la decisión tomada. De esta forma, el sistema utiliza la información que tiene para producir mejores resultados, es decir, que vayan minimizando el coste (para más información consultar [4]).

También utilizamos métodos basados en modelos; esta decisión se debe a que, como dice Schneider en el capítulo *Exploiting Model Uncertainty Estimates for Safe Dynamic Control Learning* [5], los métodos basados en modelos son más efectivos extrayendo información importante de los datos disponibles que los métodos libres o carentes de modelo, puesto que los métodos basados en modelos cuentan con un modelo previo que especifica la forma en la que se debe aprender de los datos, en contraste con los métodos libres de modelo, que elaboran un modelo propio (puesto que no cuentan con uno inicialmente), lo cual supone un proceso más costoso. Debido a los beneficios encontrados en la eficiencia de datos y velocidad de ejecución, es preferible utilizar sistemas que basados en modelos siempre y cuando las condiciones lo permitan. Esta decisión también supone un inconveniente en otros aspectos, ya que los métodos basados en modelos sufren de un sesgo por el que asumen que las dinámicas aprendidas son suficientemente buenas, cuando pueden no estar siéndolo. La aproximación de Inferencia Probabilistica para Aprendizaje de Problemas de Control (PILCO) [1] ofrece una solución a estos problemas (para más información consultar [6]).

1.1. Objetivos

Los objetivos de la realización de este trabajo son los siguientes:

- Ampliar mis conocimientos en la rama del aprendizaje automático.
- Comprender la resolución de problemas de control mediante técnicas de aprendizaje automático.
- Conocer el funcionamiento de los Procesos Gaussianos
- Aprender a utilizar herramientas actuales como TensorFlow.
- Mejorar mis conocimientos en la utilización de librerías de Python.

1.2. Conceptos Importantes

Es importante asentar algunos conceptos antes de continuar con las secciones posteriores a fin de comprender mejor las explicaciones desarrolladas en ellas.

Los tipos de problemas que se pueden resolver en el ámbito del aprendizaje automático son muy variados: procesamiento de señales, reconocimiento de patrones, análisis del lenguaje, problemas de control, etc. No obstante, podemos dividir los problemas que son capaces de resolver en dos grandes grupos, en base a los resultados que serán obtenidos:

- Problemas de clasificación: son aquellos en que, tras la introducción de un input, esperamos obtener un output acotado en una serie de etiquetas definidas específicamente.
- Problemas de regresión: se trata de problemas en que esperamos obtener una salida continua numérica, que puede ser cualquiera, sin necesidad de estar acotada entre un número concreto de posibilidades.

También es importante distinguir los problemas en base a la naturaleza de los mismos. Los problemas más comunes son aquellos en los que se alimenta a las redes con datos de la forma s:t donde s es el vector de características o vector de entrada y t el vector que identifica s en una clase o distinción concreta, pero esto no tiene por qué ser siempre así. Se distinguen dos tipos de aprendizaje:

- Supervisado: aquellos problemas en los que el conjunto de datos está definido de la forma $D = \{x_i, y_i\}_{i=1}^{N}$, donde x_i conforma la entrada del sistema e y_i el objetivo que identifica dicha entrada.
- No supervisado: son problemas en los que el conjunto de datos sigue la forma $D = \{x_i\}_{i=1}^{N}$, donde x_i corresponde a la entrada y no se cuenta con un objetivo que la identifique. Por lo tanto, los algoritmos deberán entrenarse desconociendo los patrones de salida que deben obtener.

Un concepto del que se hace uso en los capítulos relacionados con el GP es el de proceso estocástico [7]. Un proceso estocástico consiste en un conjunto de variables aleatorias y que modelan un sistema dependiente de un argumento x, de forma que para cada componente x existe una distribución probabilística asociada a y.

Hay otro concepto del que se hace uso en capítulos posteriores. Se trata del método o simulación de

Montecarlo [8]. Este método consiste en aproximar valores esperados utilizando promedios muestrales (mediante una normal de media μ y varianza σ^2) que generan muestras de variables pseudoaleatorias [9] [10].

1.3. Motivación

Durante el tercer y cuarto curso del grado cursé algunas asignaturas que me introdujeron en el aprendizaje automático, como Inteligencia Artificial y Fundamentos de Aprendizaje Automático. Estas asignaturas despertaron interés en mi, ya que aplicaban otros conocimientos aprendidos durante la carrera al mismo tiempo que abrían la puerta a un campo de una amplitud y profundidad que desconocía; inicialmente, me pareció un campo con mucho desarrollo, aunque todavía con una larga trayectoria, pero sobre todo me parecía un campo de la informática que muestra el potencial de esta rama del conocimiento, mediante resultados sorprendentes en las disciplinas en las que se aplica.

Por otro lado, la oportunidad de realizar un trabajo en el ámbito del aprendizaje automático suponía una experiencia trabajando en esta rama mucho más profunda que las vividas anteriormente y que podía aportarme una visión más realista acerca del funcionamiento de un proyecto de este tipo. Además, me llamó la atención especialmente la orientación del trabajo hacia la resolución de un problema de control, ya que anteriormente no me había enfrentado a problemas de este tipo.

Otra de las razones por las que me pareció interesante llevar a cabo un trabajo de este tipo ampliar mi conocimiento en el ámbito del aprendizaje automático, de forma que me orientase en mi futuro en cuanto a si continuar estudiando informática de una forma más genérica o centrarme en este área en específico.

1.4. Organización del documento

Este documento está dividido principalmente en 5 apartados. En el primer apartado se hablará de las principales herramientas utilizadas en este proyecto. A continuación se explicará la idea inicial del proyecto, así como las modificaciones que han sido aplicadas a la misma. En tercer lugar, se comentará el diseño del sistema desarrollado y cómo se ha abordado el problema inicial, entrando en mayor detalle en los aspectos de implementación y dando una visión más profunda del funcionamiento del sistema. Continuará con un apartado donde se explicarán las pruebas realizadas para comprobar el funcionamiento del sistema así como los resultados obtenidos de estas. Por último, en el cuarto apartado se comentarán las conclusiones alcanzadas tras la realización de este trabajo y el trabajo futuro que se plantea tras la realización de este proyecto.

2

Procesos Gaussianos y Redes Neuronales

Durante este trabajo han sido utilizadas ciertas metodologías y herramientas propias de la rama del aprendizaje automático que serán descritas a continuación, entrando en mayor detalle en los aspectos teóricos, ya que en secciones posteriores se tratará la aplicación de estas técnicas en la implementación realizada.

2.1. Redes Neuronales

Las redes neuronales nacieron en la década de 1940 de la mano de Warren McCulloch y Walter Pitts, que idearon una red basándose en el funcionamiento de transmisión de información de las neuronas del cerebro humano. Desde ese momento ha habido una gran cantidad de aportaciones y avances en este campo hasta llegar al punto en que estamos ahora.

En cuanto a los tipos de redes neuronales, podemos distinguirlas de muchas formas, una distinción puede ser en base al camino que siguen los datos dentro de las mismas. Existen dos tipos:

- *Feedforward*: son aquellas redes en la que la información fluye desde las neuronas de la capa de entrada en dirección a las neuronas de la capa de salida, pasando por las capas ocultas si las hubiese.
- Recurrentes: en este tipo de redes, existen unas conexiones entre las capas, de forma que la información retroalimenta el sistema durante un número de épocas preestablecido.

Como se comentaba anteriormente, el Perceptrón Multicapa fue un avance importante en la teoría de redes neuronales, debido a que significaba el paso de algoritmos capaces de resolver problemas de clasificación siempre y cuando la distribución de los datos fuera separable linealmente, a redes capaces de tomar un nivel superior de abstracción y resolver problemas que no cuenten con separabilidad lineal. El potencial de estas redes está directamente relacionado con dos conceptos que combinan:

- Una estructura con capas internas: utilizar estructuras con capas internas permite a la red tomar distintos niveles de abstracción, separando los datos con mucha mayor precisión.
- Un algoritmo de retropropagación: el algoritmo de retropropagación que utilizan consiste en un método de descenso por gradiente con el objetivo de minimizar el Error Cuadratico Medio (ECM) de la salida; alternativamente, se puede utilizar una función de coste a modo de guía para la resolución del problema concreto en que se aplique

dicha función (para más información consultar [11]).

Para comprender el funcionamiento del concepto de retropropagación es importante conocer el funcionamiento general de un Perceptrón Multicapa, similar al de otros algoritmos de tipo *feedforward*. La ejecución del algoritmo consta de dos partes: entrenamiento y explotación. Durante la fase de entrenamiento, se llevan a cabo tres fases principalmente:

1.– Inicialización de los pesos correspondientes a las conexiones entre las neuronas de las distintas capas de la red.

2.– Propagación de la entrada de la red hacia delante. Durante esta fase, se calcula el valor de cada neurona mediante la aplicación de la función de transferencia al sumatorio del producto de las neuronas conectadas con esta neurona con el peso de la conexión que las une.

3.– Retropropagación del error y reajuste de pesos. Una vez se ha propagado la entrada hasta haber calculado el valor de las neuronas de salida de la red, se calcula el error comparando estos resultados con los objetivos asociados a la entrada introducida; con este error, podemos iniciar una serie de operaciones que van desde las neuronas de salida hacia las neuronas de la capa de entrada calculando el ajuste de pesos necesario para la disminución del error cometido. Finalmente, se realiza el ajuste de pesos y continúa el proceso con la introducción de un nuevo patrón.

En este caso, el aprendizaje y cálculo del error se lleva a cabo minimizando una función de coste. A diferencia de como se ha explicado anteriormete, donde el ajuste de pesos se realiza tras cada introducción de un patrón de entrenamiento, en este caso se realizará posteriormente a la introducción de patrones en la red. En primer lugar se lleva a cabo la fase de explotación de la red, guardando las operaciones realizadas, así como las predicciones obtenidas, de forma que, tras el consumo del conjunto total de datos, se calcula el error total mediante la función de coste y minimiza dicho error por descenso de gradiente ajustando los pesos de la red. En el apartado de implementación 4.2 se entrará en mayor detalle en la función de coste utilizada.

En este trabajo se ha utilizado una red neuronal multicapa con arquitectura feedforward, aplicada sobre un problema de aprendizaje supervisado de regresión, en el que a partir de una información externa se producen los controles que van a ser aplicados en el sistema a controlar a fin de generar una nueva entrada para la red. Por esta razón, antes de entrenar el sistema debe hacerse una simulación mediante la explotación de la red, produciendo predicciones que son utilizadas para generar inmediatamente nuevas entradas para la red en un tiempo de simulación especificado. Durante la fase de entrenamiento, la función de activación utilizada ha sido Leaky Rectified Linear Unit (Leaky ReLU) , una modificación de la clásica función Rectified Linear Unit (ReLU) en la que se permite el paso de valores negativos pequeños a través de la función de transferencia, a diferencia de la función original ReLU, en la que se evitan los valores negativos (Ecuación 2.1a) (para más información consultar [12] [13]).

Una de las razones por las que se utiliza esta función de transferencia es la de introducir no linealidades dentro del sistema, pero la razón principal por la que Leaky ReLU es preferible en esta ocasión en lugar de ReLU, es que, en la función de transferencia de ReLU (Ecuación 2.1b), los valores

negativos devuelven 0, por lo tanto, en estos casos no hay cambio de pesos. Ambas funciones se pueden observar en la figura 2.1. Tener un cambio de pesos, aunque sea muy pequeño, beneficia el aprendizaje y evita el estancamiento.

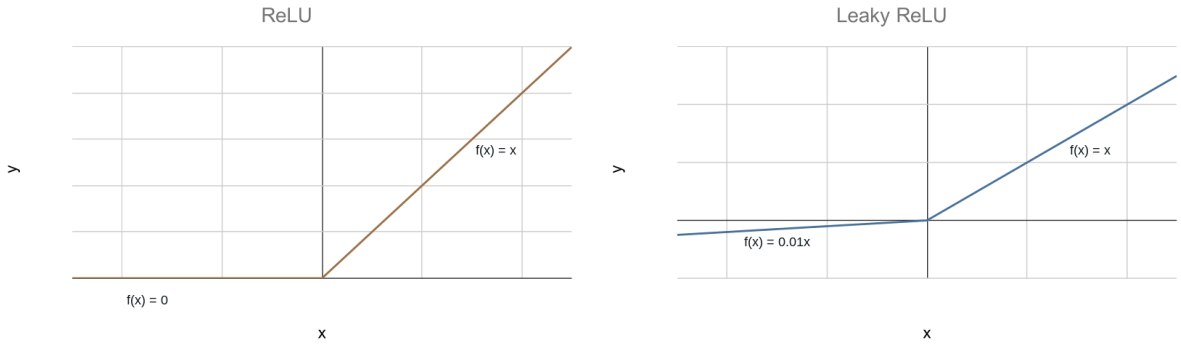


Figura 2.1: Gráficas representantes de la función ReLU y Leaky ReLU.

$$f(x) = \begin{cases} x & \text{si } x > 0 \\ 0{,}01x & \text{en otro caso} \end{cases} \tag{2.1a}$$

$$f(x) = \begin{cases} x & \text{si } x > 0 \\ 0 & \text{en otro caso} \end{cases} \tag{2.1b}$$

La red neuronal que utiliza el sistema puede contar con capas ocultas y tiene una estructura como la de la figura B.1, que es la red neuronal que ha sido utilizada durante las pruebas, aunque podría utilizarse cualquier otra que tuviera en común el número de entradas y salidas.

En la red definida, la salida o output se calcula por medio de la propagación de la entrada a lo largo de la misma. La forma en la que se ha decidido llevar a cabo es mediante el guardado de la matriz de pesos de las conexiones de la red, en la que quedan expresados todos los pesos correspondientes a las conexiones de cada capa con la siguiente.

Como se ha comentado antes, la red neuronal elegida utiliza la función de transferencia Leaky ReLU, es decir, que para calcular la salida de la red, se deben multiplicar los pesos con los valores de entrada; posteriormente se suma el valor del sesgo de cada capa, y se introduce el valor obtenido en la función de transferencia para hallar el valor de la neurona correspondiente en cada momento. Esta operación se realiza en todas las capas salvo en la última, en la que se calcula el resultado final de la red sin utilizar la función de transferencia.

La retropropagación de la red se calcula mediante el uso del optimizador Adam aplicado a los pesos de las conexiones de la red. Este algoritmo realiza el ajuste de los pesos en base a la minimización de una función de coste especificada en su versión ruidosa, mediante gradientes ruidosos. En este caso se trata de la función de coste, que calcula el coste total de todos los resultados de la red tras

introducir un conjunto de valores de entrada. Durante el capítulo 4, se entrará en mayor detalle en el funcionamiento del mismo.

2.2. Procesos Gaussianos

Los Procesos Gaussianos son un concepto clave en este trabajo. Se trata de una herramienta dentro del modelado Bayesiano, que aunque ha tenido menos atención que sus alternativas, ofrece características muy interesantes para la resolución de cualquier tipo de problema, ya sea de regresión o clasificación. En este caso, el trabajo está centrado en un problema de regresión.

Frente a otros métodos de aprendizaje automático, se ha elegido utilizar un Proceso Gaussiano en este trabajo porque, a diferencia de métodos más comunes como las redes neuronales, en los que tras introducir una entrada únicamente obtenemos una predicción, los GPs devuelven una distribución predictiva en la que se tiene en cuenta la incertidumbre del método.

En la introducción a su libro [14], Rasmussen hacía un acercamiento a este tipo de modelado que me parece importante comentar, por la claridad que aporta al desarrollo de este concepto.

En la resolución de un problema de aprendizaje supervisado en que se tienen unos datos de entrada x con sus respectivos objetivos y, se busca una función $f(x)$ con la que podamos obtener de unos valores de test x^*, sus respectivos valores objetivo. Existe una gran cantidad de funciones igualmente válidas que son capaces de realizar esta tarea. Rasmussen plantea dos posibles aproximaciones para resolver este escenario:

1.– Restringir las funciones aceptadas a un tipo concreto.

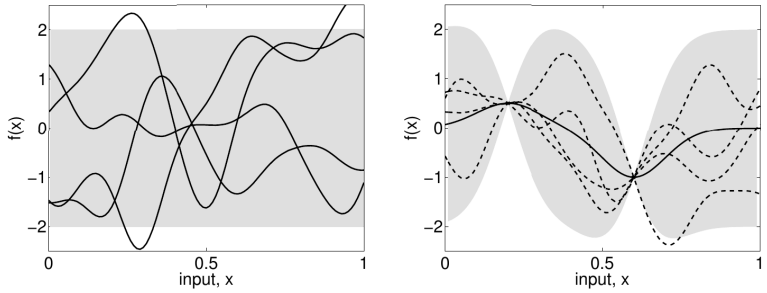
2.– Dar una probabilidad prior a todas las funciones posibles, donde mayores probabilidades van asociadas a funciones que se adecúen más, en base a distintos criterios que establezcamos.

La primera opción tiene un problema básico: tenemos que elegir en un espacio muy limitado de funciones, condicionando la solución que obtendremos a la decisión inicial de qué tipo de funciones permitir. La segunda opción también parece tener un problema claro: debemos computar un número infinito de funciones en un tiempo finito. Para resolver este problema, vamos a utilizar Procesos Gaussianos.

Un Proceso Gaussiano es una distribución de probabilidad sobre funciones en la que, tras introducir un conjunto finito de puntos, se obtiene una distribución de Gauss o distribución normal.

Inicialmente, el Proceso Gaussiano define una distribución de priores con los datos de entrenamiento (x, y). Conforme va recibiendo datos reales, esta distribución de priores (en la que están contempladas todas las funciones posibles), va convirtiendo los priores en posteriores y ajustándose a los datos reales, desechando aquellas funciones que no pasen cerca de los puntos marcados. Esto se puede ver fácilmente en la figura 1.1 del libro Gaussian Processes for Machine Learging de Rasmussen [14],

figura 2.2.



(a), prior (b), posterior

Figura 2.2: Figura extraída del libro de Gaussian Processes for Machine Learning, en la que se muestra la gráfica de la distribución de priores (a) y la distribución de posteriores una vez conocidos dos valores (b). (Permiso de utilización de esta figura concedido por la editorial MIT Press)

La distribución predictiva de un GP para problemas de regresión con observaciones ruidosas viene definida por una Gaussiana multivariante con media $K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1} y$ y varianza $K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1} K(X, X_*)$ donde y es el vector de observaciones de entrenamiento, $\sigma_n^2 I$ el ruido Gaussiano y dados n puntos de entrada y n_* puntos de test, se define la matriz de covarianzas $K(X, X_*)$ utilizando los puntos n correspondientes al conjunto de entrenamiento y n_* correspondientes al conjunto de test; esta aplicación de la función de covarianza en los distintos puntos n y n_* también se aplica en la obtención de las demás matrices de covarianzas $K(X, X)$, $K(X_*, X_*)$ y $K(X_*, X)$. La expresión de la distribución predictiva del GP se muestra en la ecuación 2.2 (para más información consultar [15] [16] [17]).

$$f_*|X_*, X, f \sim \mathcal{N}(K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1} y, \\ K(X_*, X_*) - K(X_*, X)[K(X, X) + \sigma_n^2 I]^{-1} K(X, X_*)) \tag{2.2}$$

Otro concepto importante es el de probabilidad marginal o evidencia, que corresponde a la marginalización sobre los valores de función f. La ecuación del logaritmo de la verosimilitud marginal es la mostrada en la ecuación 2.3, donde $y \sim \mathcal{N}(0, K + \sigma_n^2 I)$. Mediante esta expresión se ajustan los hiperparámetros del GP situados en la función de covarianzas, como la escala longitudinal l, según la cuál se permite la entrada de funciones de mayor o menor amplitud en la distribución predictiva o la varianza del ruido Gaussiano $\sigma_n^2 I$.

$$\log p(y|X) = -\frac{1}{2} y^T K^{-1} y - \frac{1}{2} \log |K| - \frac{n}{2} \log 2\pi \tag{2.3}$$

Llegados a este punto, parece que el Proceso Gaussiano sea un modelo que no va a requerir mayor atención en cuanto a la optimización; no obstante, se pueden modificar características del sistema en la función de covarianzas asociada al GP.

En las ecuación 2.2, la función de covarianzas o función *Kernel* utilizada es de tipo *squared exponential* [18] con observaciones ruidosas, es decir, sigue la ecuación 2.4, donde l es la escala de longitud, σ^2 la varianza y $\sigma_n^2 I$ la covarianza del ruido; ambos deben ser mayores que 0. Esta función es la que se suele tomar por defecto en Procesos Gaussianos y Maquinas de vectores de soporte o Support Vector Machine (SVMs) .

$$k(x, x_*) = \sigma^2 exp(-\frac{(x - x_*)^2}{2l^2}) + \delta_{ij}\sigma_n^2 I \tag{2.4}$$

La variación de l, σ^2 y $\sigma_n^2 I$ tiene una repercusión directa sobre los priores y la distribución predictiva del GP como se puede ver en los siguientes ejemplos [19]. En la figura B.2 podemos observar 3 muestras del prior utilizando un valor de la escala de longitud l bajo en el gráfico de la parte izquierda y utilizando un valor alto en el gráfico situado a la derecha; como se puede observar, este parámetro influye directamente en la suavidad de la distribución, es decir, la facilidad con la que los valores de las funciones pueden cambiar. Esto se ve claramente si nos fijamos en el eje y; en el caso de utilizar valores bajos de l, las tres muestras toman valores entre 4 y -4, produciéndose variaciones muy pequeñas; sin embargo, al utilizar valores altos de l, las muestras varían entre 30 y -40.

En la figura B.3 se observan otros dos ejemplos, el que está situado en la parte izquierda tiene una amplitud σ^2 muy pequeña mientras que el de la parte derecha tiene un valor alto de la misma. La amplitud influye en la variación de valores respecto a la media de los mismos; con valores mayores de amplitud, los valores de las funciones priores estarán comprendidos entre $\pm sqrt(amplitud)$. En el caso del gráfico situado a la izquierda, utilizando un valor bajo de amplitud, los valores oscilan entre 4 y -4; mientras que en el gráfico de la derecha, con un valor grande de la amplitud, oscilan entre 30 y -10.

Por último, en la figura B.4 se observa un ejemplo de la distribución predictiva del GP utilizando un valor bajo de ruido $\sigma_n^2 I$ en primer lugar y en segundo lugar, un valor muy alto del mismo. Como se puede observar, utilizar valores altos de ruido repercute directamente sobre la confianza en las observaciones realizadas. Al utilizar valores bajos de ruido en el sistema, la confianza del mismo en las observaciones realizadas crece, mientras que utilizando valores altos de ruido, el GP aumenta su incertidumbre.

Una de las principales ventajas de los Procesos Gaussianos y por la que es elegido para resolver el problema propuesto en el documento de investigación PILCO [1] es que, como no se trata de un modelo paramétrico, hay mayor flexibilidad en las funciones que definen la distribución. Además, los resultados obtenidos son bastante correctos, como cabe esperar por un modelo que unifica aspectos más sofisticados de un modelo matemático con la posibilidad de desarrollarlo computacionalmente.

3

Inferencia Probabilística para Aprendizaje de Problemas de Control (PILCO)

Este trabajo se basa en el documento de investigación PILCO [1], en el que se explica una aproximación basada en modelos para resolver problemas de tipo *policy search* de una forma óptima.

Este tipo de problemas suele aplicarse en el área de la robótica y control de sistemas interactivos con el medio. El principal objetivo de los métodos de *policy search* es encontrar las dinámicas o normas que rigen un sistema, a fin de mejorar el control que se tiene sobre el mismo [20].

Una parte importante de estos métodos es definir el objetivo que debe alcanzar el sistema controlado. Esto se suele abordar mediante la definición de una función de coste o función objetivo, en la que se evalúan los resultados obtenidos tras aplicar unos controles específicos. De esta forma, la optimización del sistema se realizará tratando de minimizar el valor de esta función mediante descenso por gradiente. Cuando se encuentra el valor óptimo definido por la función objetivo, es cuando podemos determinar que el sistema ha aprendido las dinámicas buscadas.

El principal objetivo del método de aprendizaje es reducir el sesgo producido por la utilización de un método basado en modelos, evitando así las desventajas de ineficiencia y consumo de datos de los métodos que carecen de modelos.

En las secciones siguientes de este capítulo será explicado el documento de investigación PILCO [1], en el que se detalla el funcionamiento íntegro del sistema y las pruebas realizadas; así como la aplicación efectuada de las ideas plasmadas en el documento de investigación durante el desarrollo de este trabajo y los cambios realizados respecto a la definición inicial.

3.1. PILCO: Una Aproximación Basada en Modelos para la Resolución de Problemas de Control

En este capítulo del trabajo, se procede a la explicación de las ideas y algoritmos propuestos en el documento de investigación PILCO [1] escrito por Marc Peter Deisenroth y Carl Edward Rasmussen. Esta sección está dividida en tres subsecciones, en las que se abordarán aspectos teóricos y funcionamiento del sistema propuesto, pruebas realizadas utilizando dicho sistema, y por último, conclusiones

alcanzadas tras haber comprendido la materia que expone el documento.

3.1.1. Diseño e Implementación

El sistema propuesto está formado principalmente por tres partes:

- El entorno en que se encuentra el objeto que se quiere controlar, junto con el objetivo final que se quiere lograr.
- El controlador, encargado de aprender a manejar el objeto controlado alcanzanzando el objetivo definido.
- El método que modela el sistema, aprendiendo las dinámicas del sistema y con ello, pudiendo simular el entorno de forma artificial.

El objetivo final del sistema es lograr que el controlador aprenda a manejar el objeto a controlar minimizando el número de datos requerido de simulaciones en el entorno real y reduciendo de esta forma, el impacto del controlador sobre el entorno, aumentando la velocidad de aprendizaje. Esto se va a lograr mediante la utilización de un proceso estocástico, con el que se modelarán las dinámicas del sistema, de forma que imite el entorno y se puedan hacer simulaciones artificiales en las que el controlador aprenda de forma efectiva. Por lo tanto, en primer lugar, se debe ajustar dicho modelo de forma que el modelado del sistema se adecúe al funcionamiento real del entorno, permitiendo un alto grado de independencia del sistema frente al entorno real.

Proceso Estocástico

El modelo probabilístico que controla las dinámicas del sistema, está pensado como un GP que recibe tuplas $(u_{t-1}, x_{t-1}) \in \mathbb{R}^{D+F}$ como entrada, donde x_{t-1} es el estado del sistema en un instante $t-1$ y u_{t-1} el control aplicado en dicho instante de tiempo; mientras que $\Delta_t = x_t - x_{t-1} + \varepsilon \in \mathbb{R}^D$, donde $\varepsilon \in \mathbb{R}^D$, $\varepsilon \sim \mathcal{N}(0, \Sigma_\varepsilon)$, $\Sigma_\varepsilon = diag([\sigma_{\varepsilon_1}, ..., \sigma_{\varepsilon_D}])$ son los valores objetivos o targets correspondientes a los valores de entrada.

El objetivo del GP es modelar las dinámicas del sistema, es decir, predecir el estado x_{t+1} dado un par (u_t, x_t). Para la obtención de los datos que consume el GP es necesario simular el sistema de forma aleatoria, ya que en la fase inicial, el controlador no tiene ninguna información previa acerca de cómo realizar su tarea; como se explicará en la sección 3.2, este es uno de los aspectos que han sido modificados en la realización del trabajo. Para obtener las predicciones de x_{t+1}, debemos calcular las medias μ_u y covarianza Σ_u de la distribución correspondiente a los controles aplicados u_t a fin de obtener la covarianza cruzada $cov[x_t, u_t]$ con la que se realiza el aprendizaje del GP, mediante la optimización de los parámetros de la misma.

El GP devuelve predicciones mediante la definición de la distribución de probabilidad como se muestra en la ecuación 3.1a, donde μ_t es la media definida con la ecuación 3.1b y Σ_t con 3.1c.

$$p(x_t|x_{t-1}, u_{t-1}) = \mathcal{N}(x_t|\mu_t, \Sigma_t) \tag{3.1a}$$

$$\mu_t = x_{t-1} + \mathbb{E}_f \Delta_t \tag{3.1b}$$

$$\Sigma_t = var_f[\Delta_t] \tag{3.1c}$$

La distribución predictiva para un conjunto de test $\tilde{x}_*$ se define como $p(\Delta_*|\tilde{x}_*)$, con media 3.2a y varianza 3.2b. En estas ecuaciones, es importante comentar las matrices de covarianzas $K := (\tilde{x}, \tilde{x})$, $k_* := K(\tilde{x}, \tilde{x}_*)$ y $k_{**} := K(\tilde{x}_*, \tilde{x}_*)$, donde $\tilde{x}$ corresponde a los puntos de entrenamiento n y $\tilde{x}_*$ a los puntos de test n_* con lo que $\beta := (K + \sigma_\varepsilon^2 I)^{-1} y$, donde y es el vector de observaciones de entrenamiento.

$$m_f(\tilde{x}_*) = \mathbb{E}_f[\Delta_*] = K_*^T (K + \sigma_n^2 I)^{-1} y = K_*^T \beta \tag{3.2a}$$

$$\sigma_f^2(\Delta_*) = var_f[\Delta_*] = K_{**} - K_*^T (K + \sigma_\varepsilon^2 I]^{-1} K_* \tag{3.2b}$$

Controlador

Una vez han sido aprendidas las dinámicas del sistema, se pueden realizar simulaciones utilizando el controlador, de forma que los datos obtenidos en ellas sirvan para su aprendizaje.

Durante los procesos de simulación, se utiliza la distribución predictiva del GP para obtener una salida x_{t+1} dado un par (x_t, u_t) generando muestras directamente de dicha distribución; esta salida x_{t+1} se utiliza inmediatamente como efecto nuevo estado del sistema a controlar tras aplicar el control u_t en el estado x_t.

Tras cada simulación, se utilizan todos los datos generados en la función de coste y se optimiza el controlador en base a ella. Esto se realiza minimizando el resultado esperado en la función de error 3.3, donde $\mathbb{E}$ es la esperanza, aproximada mediante un proceso de propagación de distribuciones normales a partir de GPs, con lo que $\mathbb{E}$ tiene una expresión analítica y c la función de coste que se muestra en la ecuación 3.4; es una función de tipo *squared exponential*, donde x es el estado actual, x_{target} el estado objetivo y σ_c^2 se utiliza para controlar la envergadura del resultado. Para que la aproximación Gaussiana lleve a esta función, se utiliza un controlador basado en funciones de base radial. Por último, al final de cada iteración, se guardan los datos obtenidos en la simulación para una mejor optimización del sistema en las siguientes iteraciones.

$$J^\pi(\theta) = \sum_{t=0}^{T} \mathbb{E}_{x_t}[c(x_t)], x_0 \sim \mathcal{N}(\mu_0, \Sigma_0) \tag{3.3}$$

$$c(x) = 1 - exp(- \parallel x - x_{target} \parallel^2 / \sigma_c^2) \in [0, 1] \tag{3.4}$$

Algoritmo PILCO

El algoritmo que se sigue durante todo este proceso es el descrito en la figura A.1, correspondiente al Algoritmo 1 expuesto en el documento de investigación PILCO [1].

3.1.2. Pruebas y Experimentos

El documento proponen algunos experimentos con este sistema; tras estudiar dichos experimentos, se ha seleccionado el ejemplo más representativo, que será explicado en esta sección.

Se trata de un carrito de masa 0.5 kg al que está enganchado un péndulo con dos articulaciones de 0.5 kg cada una. El objetivo consiste en colocar el péndulo doble en posición invertida partiendo de un estado inicial x_0. Las variables que modelan este sistema son:

- Posición x_1.
- Velocidad $\dot{x}_1$.
- Ángulos θ_2 y θ_3.
- Velocidades angulares de las dos partes del péndulo.

Las señales de control se aplican al movimiento horizontal del carrito. El estado inicial consta con el carrito colocado en la posición x_0 y ambas articulaciones del péndulo se encuentran colgando. El estado final por tanto, consta del péndulo colocado en posición invertida en una posición x.

Una aproximación común para resolver este problema consistiría en utilizar dos controladores: uno para manejar el balanceo y otro para colocar el péndulo en posición invertida. Sin embargo, PILCO es capaz de resolver este problema únicamente utilizando un controlador de tipo Radial Basis Function (RBF) para resolver ambas tareas de control de forma conjunta entre 20 y 30 intentos, es decir, entre 60 y 90 segundos.

3.1.3. Resultados y Conclusiones

Como podemos ver por los resultados obtenidos en el ejemplo anterior, una de las mayores ventajas de PILCO es la eficiencia de datos, ya que con una cantidad muy pequeña de datos, es capaz de encontrar la solución al problema. En el gráfico B.5 que se muestra en la *Figura 5* del documento de investigación se muestra la comparación de PILCO con otros métodos de aprendizaje reforzado que tienen en común que en todos los casos aprenden sobre la marcha, es decir, sin un conocimiento previo de los datos. El problema sobre el que han sido aplicados en este gráfico ha sido *cart-pole*; se trata de un problema parecido al experimento explicado, en el que se tiene un carrito con un péndulo de una sola articulación que se quiere colocar en posición invertida, moviendo el carrito en el eje

horizontal.

Como conclusiones del documento, se comenta que PILCO, en problemas de ensayo-error, no es un método de control óptimo, ya que encuentra únicamente una solución, que no tiene por qué ser la solución óptima del problema al que se aplique. Uno de sus mayores éxitos consiste en la reducción del sesgo de utilizar un método basado en modelos, introduciendo la incertidumbre en los distintos pasos del algoritmo. PILCO define un método de aprendizaje por refuerzo vanguardista en términos de eficiencia de datos y velocidad de aprendizaje.

3.2. Aplicación y Modificaciones

Tras una comprensión del funcionamiento del sistema descrito en el documento de investigación PILCO [1], en esta sección se comentará la aplicación final de las técnicas y algoritmos descritos, así como las modificaciones llevadas a cabo de la línea principal de actuación que propone el documento.

Durante el desarrollo de este trabajo han sido necesarias algunas extensiones y modificaciones respecto a cómo está descrito el sistema originalmente en PILCO.

Uno de los primeros módulos implementados fue el correspondiente al modelo que aprende las dinámicas del sistema, es decir, el GP. En el desarrollo de este módulo no ha habido grandes diferencias en cuanto al funcionamiento del mismo, debido a que se han seguido los algoritmos y ecuaciones que forman un GP descritos en el libro *Gaussian Processes for Machine Learning* [14], escrito por uno de los autores del documento PILCO. No obstante, en dicho documento se hace incapié en en cálculo analítico del gradiente para la optimización del GP, pero durante su implementación, la optimización ha sido realizada utilizando TensorFlow mediante su cálculo automático de gradientes gracias al optimizador *Adam*, del que se hablará en capítulos posteriores. Esta decisión se debe a la complejidad de realizar esta tarea cuando ya contamos con un algoritmo que realiza correctamente ese trabajo; además, por razones de tiempo y conocimientos matemáticos requeridos, es una carga de trabajo que se desvía de los propósitos iniciales del trabajo.

En cuanto al funcionamiento del sistema PILCO, este es muy similar al descrito en el documento de investigación, a excepción de los cálculos de gradientes, por la misma razón comentada anteriormente. El algoritmo implementado en la optimización, mecanismo principal del módulo, es el mismo que se describe en el algoritmo A.1.

Otro cambio llevado a cabo es que, durante todo el trabajo se ha asumido que las entradas de test siguen una distribución Gaussiana multivariante; a diferencia del documento original, durante la realización del trabajo se ha cambiado la forma en que el GP aprende durante la fase de simulaciones aleatorias. Mientras que en la idea original se propone introducir entradas aleatorias y aproximar las salidas a una distribución Gaussiana de forma analítica, en este trabajo se ha utilizado el método

de Montecarlo para aproximar la función de coste del sistema, generando muestras de la distribución predictiva del GP. Esto se puede ver en la ecuación 3.5, donde θ son los parámetros de la red neuronal.

$$J^{\pi}(\theta) = \sum_{t=0}^{T} c(x_t), x_t \sim \mathcal{N}(\mu_0, \Sigma_0) \tag{3.5}$$

En cuanto a la función de error definida en la ecuación 3.3, también ha habido modificaciones, ya que la esperanza $\mathbb{E}$ se ha aproximado por Montecarlo, mediante la generación y posterior propagación de muestras. Esto se ha decidido de esta forma porque utilizando Montecarlo no es necesario que la esperanza tenga una solución analítica. La propia función de coste c mostrada en la ecuación 3.4 también ha sido modificada, utilizando finalmente la que se muestra en la ecuación 3.6.

$$c(x) = \| x - x_{target} \|^2 \tag{3.6}$$

Además de los cambios en la metodología que sigue el algoritmo comentados anteriormente, los cambios más importantes son aquellos relacionados con la realización de pruebas y la simulación del entorno. Esto se debe a que, como no contábamos con los bienes físicos para realizar las simulaciones que se llevan a cabo en el documento inicial, se decidió implementar también la simulación del entorno, imprescindible para probar el funcionamiento del método, así como los demás módulos.

Por tanto, el sistema que se aprende a controlar, así como el controlador que se aplica al mismo, son más simples que el carrito con un péndulo de doble articulación y el monociclo descritos en la sección anterior. En este caso, el problema consiste en mover un carrito de un punto x_0 a un punto $x_{objetivo}$ utilizando un controlador propio. Durante las pruebas realizadas se podrá ver el funcionamiento del sistema utilizando este marco que, aunque no es tan vistoso o complejo como el descrito en las pruebas del documento de referencia, utiliza el mismo sistema, y por tanto, funciona del mismo modo, aunque está aplicado a un problema distinto.

El ejemplo sobre el que ha sido desarrollado el sistema funciona de la siguiente manera: se tiene un objeto situado en una posición inicial x_0 y se quiere utilizar un controlador para mover dicho objeto a una posición objetivo $x_{objetivo}$. Para ello, aplicará un control u_0, obtenido mediante la red neuronal sobre el objeto controlado, llevándolo a la posición x_1, mediante un movimiento respectivo a u_0 unidades de espacio a partir de la posición x_0; esto se repetirá durante un número de unidades de tiempo especificado. Durante estas simulaciones, el controlador tratará de acercar el objeto a la posición $x_{objetivo}$ con el menor número de movimientos posible.

Por otra parte, se ha desarrollado un módulo que genera pares (x_t, u_t), utilizando el controlador descrito anteriormente o aplicando un control aleatorio sobre el objeto mediante una simulación sobre el entorno real durante un periodo de tiempo definido. Con ello se obtienen datos que muestran el

comportamiento del objeto respecto al entorno al aplicar determinados controles.

Estos datos obtenidos de simulaciones mediante dicho módulo son consumidos por el GP a fin de aprender las dinámicas del sistema y predecir cuál será la posición x_{t+1} para cada una de las posiciones x_t del objeto controlado. En todo momento, el encargado de gestionar estos otros módulos siguiendo el algoritmo A.1 es el módulo Pilco, en el que se introducen los parámetros relativos a la duración de las simulaciones y del aprendizaje del sistema sobre las mismas.

4

Implementación en TensorFlow

Durante el desarrollo del trabajo se ha utilizado Python3.7. Este lenguaje es mundialmente utilizado en las distintas áreas del aprendizaje automático debido a su legibilidad y la calidad de las librerías disponibles, que le dan un fuerte apoyo a distintas ramas. Este lenguaje también soporta orientación a objetos, lo cual ha permitido una mejor organización del código, facilitando también la utilización de la librería desarrollada. Podemos destacar el uso de ciertas librerías o extensiones importantes que han sido utilizadas:

- **Numpy 1.18.1**. Se trata de una librería de funciones matemáticas que también facilita las operaciones sobre vectores y matrices, así como la utilización del factor aleatorio.
- **TensorFlow 1.15.0**. Es una librería desarrollada para tareas de aprendizaje automático. Su mayor ventaja es la optimización sobre la ejecución que nos proporciona, así como las distintas operaciones matemáticas para manejo de matrices, que nos permiten aprovechar en gran medida el potencial de este framework.
- **Matplotlib 3.1.3**. Es una librería para la creación de gráficos y animaciones de distintos tipos. Nos permite representar los resultados obtenidos en el trabajo de forma automática sin tener que exportar los datos obtenidos en las pruebas y realizar este trabajo de una manera mucho menos práctica.

Antes de comenzar creo que es importante hacer una serie de comentarios acerca del funcionamiento de TensorFlow, ya que tiene un funcionamiento y una forma de utilizarse que es importante comprender a fin de entender mejor el desarrollo del trabajo.

TensorFlow es una librería que organiza las tareas en un grafo computacional a fin de optimizar la ejecución final, reduciendo las repeticiones de ejecución de ciertas operaciones. Esto es, cuando se declara una expresión de TensorFlow, esta expresión es introducida en el grafo computacional en forma de Tensor, de forma que, cuando más tarde se quiera obtener el resultado de la ejecución de esta operación, se realizará una llamada el método `run` de la sesión de TensorFlow en la que se esté trabajando en ese momento. El concepto de sesión es importante, ya que gracias a ella podemos ejecutar las operaciones de TensorFlow; también contiene el valor de las variables, que permite inicializar manualmente, lo que se debe hacerse siempre antes de ejecutar Tensores que dependan de ellas.

Un ejemplo del grafo computacional que se genera al escribir una expresión en TensorFlow es el que se puede observar en la Figura 4.1. En esta figura se representa la operación $A^2 + 2A + B$, en la que A y B son variables y el número 2 que multiplica a A, es una constante.

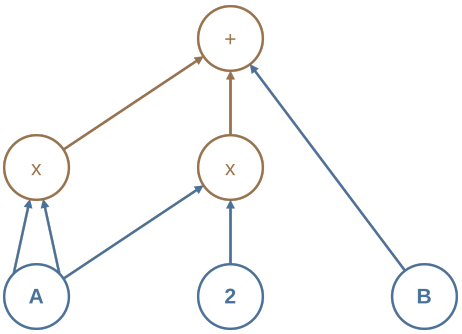


Figura 4.1: Figura correspondiente al grafo computacional creado en TensorFlow para realizar la operación $A^2 + 2A + B$.

TensorFlow utiliza algunos tipos de datos que también conviene comentar. Para abreviar, a continuación se explican los tipos de datos más importantes que han sido utilizados durante el trabajo.

- **PlaceHolders**: Cuando se declara una operación de TensorFlow, es necesario introducir en ella los operandos. En caso de no contar con el operando que se va a utilizar en esta operación o bien se tienen planes de ejecutar esta operación variando el valor de los mismos, se introducen en ella PlaceHolders, que ocuparan este espacio necesario para la declaración de la operación, pero no tienen ningún valor en el momento de la declaración.Este valor será introducido en el momento de la ejecución, de forma que la operación pueda realizarse correctamente.
- **Variables**: Se trata de una figura similar a la de los PlaceHolders, ya que en el momento de la declaración también carece de valor, pero en este caso debe quedar indicado el valor que va a recibir. La introducción de valores en las variables no se realiza en el momento de la ejecución, sino que se debe llamar a una directiva específica de la sesión en que se está operando en ese momento, que inicializa las variables de la sesión con los valores de la declaración. Al igual que los PlaceHolders, su valor puede cambiar.

Otra utilidad importante de la que se hace uso durante la implementación es la optimización de variables por gradientes. Para computar gradientes, TensorFlow utiliza diferenciación automática: un método para llevar a cabo la derivada de una función [21]. Para ello, TensorFlow almacena las operaciones realizadas, de forma que cuando se vayan a calcular el gradiente, se puedan realizar estas operaciones de forma inversa. Este mecanismo es especialmente útil en operaciones como la retropropagación de redes neuronales o maximización de la probabilidad marginal en el GP.

En este caso, el optimizador utilizado ha sido Adam [22], que es uno de los algoritmos más utilizados en este tipo de problemas de los declarados en TensorFlow [23]. Se trata de un algoritmo de optimización estocástica que trabaja con gradientes ruidosos en lugar de utilizar gradientes exactos, y tiene algunas ventajas como su eficiencia computacional o el bajo uso de memoria [24].

4.1. Procesos Gaussianos

Se trata del módulo principal del proyecto. Está codificado en el archivo *GP.py*. En él se hace referencia a TensorFlow, Numpy y RBF_ARD. Esta última librería se trata de un módulo cedido por mi

tutor, en el que se encuentra el funcionamiento relativo a las funciones *Kernel* que utiliza el Gaussian Process (GP) .

El funcionamiento del GP es el siguiente: en primer lugar, se inicializan las variables del módulo, así como la función de covarianza del GP. A continuación, se introducen en él los datos de entrenamiento, con los que modelará la dinámica del sistema en que se utilice, para finalmente obtener la expresión de las medias y varianzas que definen la distribución predictiva del GP, ecuación 2.2.

4.1.1. Inicialización

Durante la incialización se crean una serie de recursos que son importantes en el resto de la implementación como es el caso de los PlaceHolders (Código C.1) o la declaración de la función de covarianzas.

También es importante introducir los los datos de entrada a fin de extraer de ellos algunas métricas que son de utilidad para la declaración de la función de covarianzas. Además, se hace una estandarización de estos datos, a fin de evitar problemas originados por la extensión del conjunto de datos.

4.1.2. Entrenamiento

En el entrenamiento (método `train`) se hacen una serie de comprobaciones iniciales acerca de los datos de train, ya que no es necesario que el usuario introduzca las entradas de entrenamiento al llamar a este método, pues ya los ha tenido que introducir durante la inicialización; en caso de introducirlos por otras razones, se debe hacer el mismo tratamiento a estas entradas que se hacía en la inicialización a fin que la forma de los datos se corresponda a la que esperan las expresiones declaradas.

A continuación se redefinen las funciones *Kernel* $K(X, X)$, $K(X_*, X)$, $K(X, X_*)$ y $K(X_*, X_*)$ en base a los datos de entrenamiento n o test n_* según corresponda, así como la expresión de la verosimilitud marginal siguiendo la ecuación 2.3 y el optimizador Adam, que es utilizado en esta ocasión para el aprendizaje del GP, ajustando los parámetros de la función de covarianzas mediante la maximización del logaritmo de la verosimilitud marginal.

4.1.3. Fase de Predicción

El método correspondiente a esta fase es `predict`, en el que son ejecutadas las medias y varianzas de la distribución predictiva; definidas en los métodos `means` y `varianzes`.

Estos métodos utilizan las definiciones de medias y varianzas que definen la distribución predictiva mostradas en la ecuación 2.2 con sus respectivas funciones *Kernel*, definidas con los datos de test que

entran en la función y con los datos de entrenamiento que han sido guardados durante la inicialización del GP, aunque si ha habido cambios en los datos de entrenamiento, también pueden ser introducidos en esta función, ignorando los datos iniciales. Lo que deben tener en común estos nuevos datos de entrenamiento con los datos introducidos inicialmente en la inicialización del GP es la forma, pues el proceso estocástico está creado para manejar datos que tengan la forma de los datos introducidos en la inicialización del módulo.

4.2. Redes Neuronales

Se ha desarrollado una red neuronal cuyo rol es el de controlador del objeto correspondiente. Mediante este módulo se obtiene el control a aplicar dado un estado del objeto en un instante concreto, de forma que se decida el movimiento a realizar para ir acercando dicho objeto controlado al estado objetivo, a fin de alcanzarlo. Es utilizado por el módulo ControlProblem a fin de generar datos que consuma el GP.

Como se ha comentado en el capítulo 2.1, se trata de un Perceptrón Multicapa con restropropagación utilizando una función de coste. Esta función de coste se aplica una vez se conocen todas las predicciones de la red en un periodo de tiempo concreto.

La implementación de la red neuronal ha sido llevada a cabo guardando los pesos de las conexiónes entre las distintas conexiones de la red en forma de matriz de pesos en la que cada peso es una variable de TensorFlow inicializada aleatoriamente mediante la generación de muestras de una distribución normal y la reducción de este *sample* a un valor pequeño. La dimensión de esta matriz de pesos viene definida por la estructura de red que recibe el módulo como argumento durante la inicialización; así se puede ver en el fragmento de código C.2.

4.2.1. Inicialización

Durante la inicialización se crea la estructura de red inicializando el valor de las conexiones con valores aleatorios pequeños mediante la utilización de Variables de TensorFlow como se puede observar en el Código C.2.

También se crea un PlaceHolder para almacenar los inputs de test, a fin de declarar a continuación la expresión de las predicciones (Código C.3).

4.2.2. Predicción

Han sido declarados dos métodos:

1.– `predict`: se declara la expresión de las predicciones en base a los datos de test que recibe por argumento. Para el cálculo del valor de cada capa, en primer lugar se calcula la suma pesada de entradas con el producto entre el valor de las neuronas de la capa anterior y los pesos de las conexiones que las unen a la capa correspondiente. Además, se realiza la suma del sesgo de la capa correspondiente, con lo que obtenemos el valor a introducir en la función de transferencia Leaky ReLU de las neuronas de la capa que estamos evaluando. Este proceso se realiza con todas las capas de la red empezando con la capa de entrada y en dirección a la capa de salida, ya que se trata de una red *feedforward*. Por último, en la capa de salida, se evita utilizar la función de transferencia, pero se haya en su lugar la tangente hiperbólica, a fin de suavizar los resultados obtenidos de la red.

2.– `get_predictions`: se encarga de la ejecución de la función de predicciones creada durante la inicialización del módulo. Para ello, rellena el valor del PlaceHolder correspondiente a los valores de test con los valores introducidos en la misma.

4.2.3. Entrenamiento

El entrenamiento de la red se realiza de forma externa a este módulo, ya que se tiene acceso a las conexiones de la red, que son las variables que cambian en esta fase a fin de obtener un output más certero. El entrenamiento consiste en la optimización de los pesos de la red minimizando la función de coste (ecuación 3.4) definida en el módulo en que se realiza esta tarea (*Pilco*). Este entrenamiento consta de dos partes:

- Optimización por el método de Montecarlo: se toman valores aleatorios para optimizar la red durante la primera ronda completa de la optimización de PILCO.
- Durante el resto de rondas, se realiza una optimización estocástica utilizando gradientes ruidosos en lugar de exactos, aprovechando una de las principales ventajas de utilizar Adam en lugar de otro optimizador.

Durante la fase inicial de ejecución de la red, se calcula la expresión de TensorFlow correspondiente a las predicciones de la red. De esta forma, cuando se minimiza el error mediante la función de coste, se toma esta expresión de TensorFlow y se calculan los nuevos pesos mediante descenso por gradiente utilizando el optimizador Adam (para más información consultar [25]).

4.3. Problema de Control

El problema de control, codificado en el módulo ControlProblem (*ControlProblem.py*), es el encargado de la generación de datos mediante la simulación del problema, haciendo uso de al red neuronal descrita anteriormente, así como de la evaluación de la calidad de un estado en concreto mediante la función *fitness*. Como en los módulos anteriores, también utiliza Numpy y TensorFlow por motivos de cohesión entre las distintas partes del código.

Su principal utilidad reside por tanto, en los métodos de simulación, en los que se producen datos de la forma *(posición x_t, movimiento aplicado u_t)* de distintas formas, que serán utilizados posteriormente

en el módulo Pilco.

4.3.1. Inicialización

Como no podía ser de otra forma, el problema de control debe recibir el controlador para poder simular los movimientos u_t que serán aplicados dada una posición x_t. En caso de no introducir un controlador, se creará el controlador desarrollado como valor por defecto. También es necesario que reciba la posición objetivo a fin de poder calcular el fitness correctamente, cuya operación será descrita a continuación.

4.3.2. Fitness

La función fitness es la encargada de evaluar la cercanía del carrito a la posición final. En este caso, dado que el dato principal es el punto en que se encuentra el carrito, la función fitness es aplicada sobre la posición relativa que ocupa respecto a la posición objetivo en un instante de tiempo t, siguiendo la ecuación 4.1:

$$Fitness_t = (x_t - x_{objetivo})^2 \tag{4.1}$$

4.3.3. Simulación

Como se ha comentado anteriormente, la simulación es la funcionalidad principal de este módulo. En su implementación, se hace referencia a otros métodos del módulo utilizados para hacer operaciones intermedias:

- `generate_position`: genera una posición aleatoria generando muestras de una distribución uniforme entre los límites establecidos. Permite elegir si esta operación quiere obtenerse como un resultado utilizando la librería Numpy o como una expresión de TensorFlow.
- `move`: dada una posición x y un control u, aplica este control a la posición obteniendo la nueva posición alcanzada.

La forma de ejecutar simulaciones desde fuera del módulo es llamar al método `simulate`, al que se debe especificar el tiempo total en que se quiere simular el problema, así como si quiere que se aplique un control aleatorio a la hora de hacer los movimientos o bien utilizar el controlador. En base a este último parámetro, el método seleccionará entre los métodos `simulate_random_control` y `simulate_applying_control`.

Ambos métodos comparten una estructura común, diferenciándose principalmente en el tipo de

controlador que es utilizado para obtener los movimientos. Los pasos que se siguen en la estructura común se pueden observar en el Algoritmo A.2.

4.4. Pilco

En este módulo unifica la funcionalidad desarrollada en los demás módulos comentados anteriormente. Se encarga de aplicar el algoritmo descrito en el documento de investigación PILCO [1] para la resolución del problema, utilizando cada módulo como un rol concreto de la siguiente manera:

- **Red Neuronal (NNetwork)**: Como el controlador encargado de expedir un control a aplicar dada una posición a fin de minimizar la lejanía del carrito respecto a la posición objetivo.
- **Problema de Control (ControlProblem)**: Desempeña el rol de generador de datos y comprobación de la calidad de los mismos mediante el *fitness*.
- **Gaussian Process (GP)**: Aprende las dinámicas del sistema haciendo uso de los datos generados por el problema de control.

4.4.1. Inicialización

Se inicializan las variables de clase correspondientes a los valores directos de los argumentos de entrada, como son el controlador, el problema de control y el GP, además de los siguientes parámetros importantes:

- **epochs**: Limita el número de épocas de optimización de los pesos de la red neuronal realizadas en base al coste.
- **opt_rounds**: Limita el número de rondas completas de optimización; esto es, el número de ciclos completos de entrenamiento del controlador, simulación del problema y entrenamiento del GP.

4.4.2. Entrenamiento y ajuste del GP

Se lleva a cabo en el método `fit_GP`, que recibe las entradas del GP de la forma $D = \{(x_t, u_t)\}_{t=0}^{N}$ y sus objetivos o targets $D_{objetivo} = \{x_{t+1}\}_{t=0}^{N}$ sin normalizar. En este método se realiza una estandarización tanto de las entradas como de los objetivos de los ejemplos que entran por argumento. Esta estandarización se hace siguiendo la ecuación 4.2, donde x es el dato concreto a estandarizar, es decir, la coordenada del vector de entrada o del vector de salida correspondiente, m_f la media y σ_f^2 la desviación estándar asociadas al atributo correspondiente.

$$x_{normalized} = \frac{x - m_f(x)}{\sigma_f^2(x)} \tag{4.2}$$

Además de realizar la estandarización, se guardan las medias y desviaciones estándar tanto de las entradas como de los valores objetivo a fin de aplicar los mismos factores de estandarización cuando testeemos el sistema. Por último, se entrena el GP con los nuevos datos obtenidos.

4.4.3. Obtención del coste

La obtención del coste consiste en el cálculo de la media de costes de cada una de las posiciones alcanzadas en una simulación obteniendo los controles a aplicar del controlador y aplicando el GP durante un tiempo `total_time`; finalmente, se realiza la función mostrada en la ecuación 4.3, donde n es el número de unidades de tiempo en que se ha realizado la simulación y x la lista de posiciones alcanzadas durante la simulación. En ella que se calcula la suma total de las posiciones alcanzadas aplicando los controles obtenidos por la red neuronal. Este método (`get_cost`) devuelve la expresión de TensorFlow referente a las operaciones descritas anteriormente, pues será utilizada posteriormente en la optimización de los pesos del controlador.

$$coste = \sum_{i=0}^{n} fitness(x_i) \tag{4.3}$$

4.4.4. Optimización

Codificada en el método `optimize`, la optimización hace uso de varias herramientas anteriores a fin de optimizar el controlador para obtener movimientos mejores y entrena el GP para mejorar sus predicciones. Para una mejor comprensión del algoritmo que lleva a cabo, se muestra el procedimiento en el Código C.4).

5

Pruebas y resultados

Durante el desarrollo del trabajo han sido elaboradas una serie de pruebas, tanto para comprobar el funcionamiento del GP como como del sistema PILCO. A continuación serán explicadas estas pruebas así como el sentido que tienen y los resultados obtenidos en las mismas.

5.1. Tests sobre Procesos Gaussianos

El primer módulo desarrollado y pieza angular del trabajo fue el GP . Como se ha comentado en capítulos anteriores, es el encargado del aprendizaje de las dinámicas del sistema, con lo que es importante que funcione correctamente a fin de asegurar el funcionamiento del resto de módulos.

5.1.1. Funcionamiento

El funcionamiento del test consiste en la creación de unos conjuntos de datos de entrenamiento y test con una forma concreta: entradas y salidas, definidas como vectores de dimensión 1. Esto se ha decidido así a fin de poder representar los resultados de una forma más clara. El número de datos de entrenamiento y test es especificado mediante los parámetros `trainSize` y `testSize`. La variación de estos parámetros nos dará una visión más certera de cómo funciona el GP, como veremos a continuación.

5.1.2. Implementación

Es importante recalcar que los valores objetivos son generados de la siguiente forma: en primer lugar se generan datos de forma aleatoria entre los límites $[-5, 5]$; a continuación se les aplica la función seno; por último, se suma el valor de una distribución normal de la misma dimensión del conjunto. Esto se hace para modelar los valores objetivo del sistema a fin que sigan una lógica. La implementación consta de los siguientes pasos:

1.– Ajuste de parámetros: se ajusta el tamaño de los conjuntos de entrenamiento y test.

2.– Generación de datos de entrenamiento y test: se generan los datos con la dimensión seleccionada y se aplica el modelado comentado anteriormente.

3.– Creación de la sesión de TensorFlow e inicialización del GP.

4.– Entrenamiento del GP.

5.– Predicción del GP sobre los datos de test: tras la ejecución de la predicción se obtienen las medias y varianzas de la distribución predictiva del GP.

6.– Creación de gráficos.

5.1.3. Resultados Obtenidos

A continuación se describen varias ejecuciones cambiando el tamaño de los conjuntos de entrenamiento y test. En cada ejecución se generan dos gráficos: el primero, situado en la parte superior, representa los datos de entrenamiento frente a sus valores objetivo; en segundo lugar, en la parte inferior, se muestran los valores del gráfico anterior como puntos, así como los valores de la media en forma de línea negra junto con la adición y resta de la varianza, representada como una línea de color rojo. Tanto la media como la varianza están aplicadas sobre los datos de test.

Destacar que, como se ha comentado en el capítulo 2.2, la media corresponde al valor medio de la distribución de funciones del GP aplicada sobre los datos de test; así como la varianza del conjunto de funciones representa los valores más lejanos de este conjunto de funciones.

En el ejemplo de ejecución mostrado en la figura 5.1 se muestra un ejemplo de ejecución estándar con 10 datos de entrenamiento y 100 datos de test. Como se puede observar, el ajuste de las funciones a los datos de entrenamiento es bastante certero, mostrando algo más de incertidumbre en los datos de test situados entre los datos conocidos de entrenamiento, aunque mayormente se mantiene una incertidumbre constante.

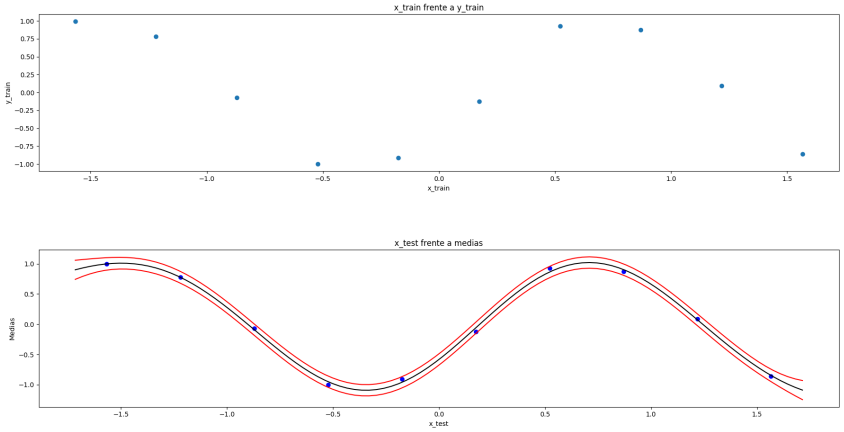


Figura 5.1: Gráfico que representa los resultados de test_GP sobre 10 datos de entrenamiento y 100 datos de test.

En el caso de la ejecución mostrada en la figura 5.2 han sido utilizados 5 datos de entrenamiento, manteniendo el mismo número de datos de test a fin de suavizar el resultado. Como se puede observar, los niveles de incertidumbre son mucho mayores en este caso, ya que, debido al escaso número de observaciones utilizadas durante la fase de entrenamiento, las funciones posibles contempladas en la distribución del GP son mucho más variadas.

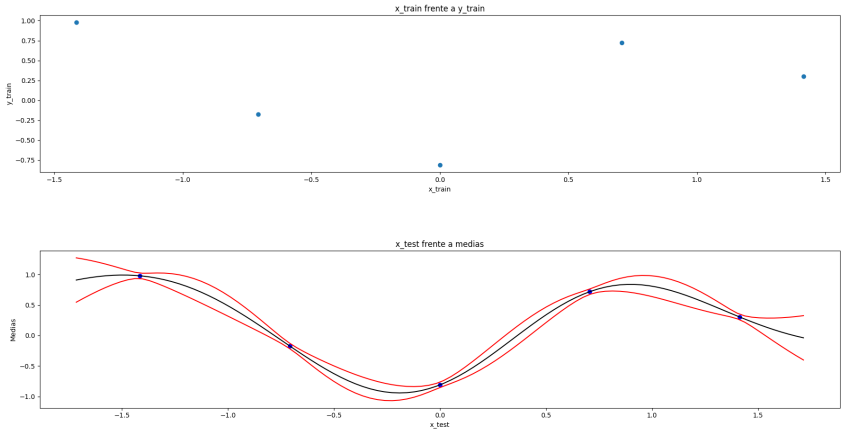


Figura 5.2: Gráfico que representa los resultados de test_GP sobre 5 datos de entrenamiento y 100 datos de test.

Por último, en la figura 5.3 se muestra una ejecución con muchas más observaciones en el conjunto de entrenamiento. Los resultados muestran cómo la media oscila en posiciones intermedias entre los valores objetivos más altos y más bajos, tratando de imitar la función que siguen estas observaciones. La incertidumbre mantiene unos valores bajos en todo momento, debido al amplio número de datos conocidos por el GP.

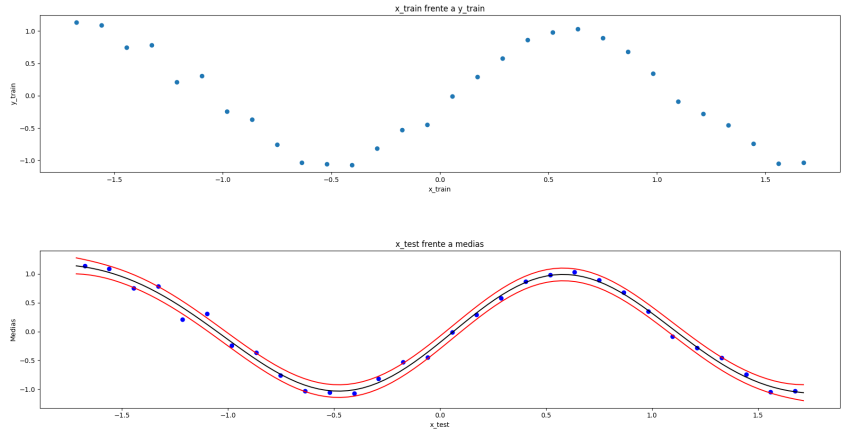


Figura 5.3: Gráfico que representa los resultados de test_GP sobre 30 datos de entrenamiento y 100 datos de test.

Tras el estudio de estas ejecuciones, se puede observar que la estimación de la incertidumbre es adecuada, aumentando en los casos en que hay muy pocas observaciones y disminuyendo conforme

se aumenta el número de observaciones conocidas.

Tras las pruebas realizadas, se han alcanzado las siguientes conclusiones: debido a su funcionamiento interno, los GPs pueden obtener predicciones basadas en una cantidad muy pequeña de datos de entrenamiento, encontrando de forma bastante efectiva el funcionamiento del sistema al que sea aplique; la utilización de una cantidad demasiado baja de datos de entrenamiento significará unos resultados con mucha incertidumbre, pero una ventaja que nos brindan los GPs es que, si interpretamos el resultado obtenido, veremos que la solución obtenida no es única, ya que hay infinitas funciones que encajan con las observaciones previas, e ir modelando el resultado con la adición de más observaciones ayudará a obtener un resultado más exacto hasta encontrar una solución con un error muy bajo.

5.2. Tests sobre PILCO

A continuación se hablará acerca del test realizado sobre el sistema PILCO. Para el desarrollo de este test es necesario tener todos los módulos del sistema funcionando correctamente, ya que van a ser necesarios para ejecutar las simulaciones. Por esta razón, nos aseguramos previamente de que el GP funcionaba bien.

5.2.1. Funcionamiento

En esta prueba se utiliza el algoritmo descrito en PILCO para resolver un problema de control sencillo: se tiene un objeto controlable en una posición inicial aleatoria x_0 y se quiere llevar dicho objeto a una posición objetivo x_{target}. Para ello, se pueden aplicar controles al objeto que harán que se mueva en el eje horizontal. Los controles u_t estarán en el rango $[-1, 1]$ y los límites del entorno se encuentran en $x = -5$ y $x = 5$. Tras aplicar un control u_t en una posición x_t, se obtendrá la siguiente posición de la forma $x_{t+1} = x_t + u_t$.

La estructura que sigue el test es similar a la que sigue *Test_GP.py*, con una fase inicial de ajuste de parámetros de la ejecución, creación e inicialización de variables que va a necesitar el sistema PILCO, y por último, ejecución de dicho sistema mediante la función `optimize` y guardado de los resultados. En este caso, los resultados son guardados como una animación en formato *gif*, en la que se muestra la progresión con el paso de las simulaciones del aprendizaje del controlador, así como un gráfico que muestra también la progresión del controlador de forma estática.

Durante esta prueba, el sistema PILCO ejecutará el método `optimize` explicado anteriormente, con lo que se llevarán a cabo épocas de entrenamiento del controlador, simulaciones para la generación de datos y entrenamiento del GP durante un número concreto de rondas especificado. La variación de estos parámetros permite la obtención de información importante acerca del funcionamiento

del sistema, que será expuesta en secciones posteriores.

5.2.2. Implementación

La implementación de la prueba es bastante simple, ya que la carga de complejidad se encuentra principalmente en que todos los componentes que interactúan en PILCO lo hagan de forma correcta. Mediante esta prueba, con un diseño sencillo, se ha podido ajustar el funcionamiento del sistema de forma que todo funcione correctamente, así como obtener los resultados que se mostrarán en la siguiente sección. El orden de ejecución de esta prueba es el siguiente:

1.– Ajuste de parámetros: se ajustan los parámetros relativos a:

- La posición objetivo (`target_position`): punto que debe alcanzar el carrito utilizando el controlador.
- El tiempo de simulación inicial (`total_time_ini`): especifica el número de unidades de tiempo de la primera simulación previa al inicio de la optimización del controlador y el GP; es decir, el número de movimientos y posiciones que se van a generar en dicha simulación. Esta primera simulación se hace utilizando un controlador aleatorio, debido a que todavía no se conoce información del entorno ni de las dinámicas que sigue el carrito.
- El tiempo de simulación durante las iteraciones de PILCO (`total_time_iter`): especifica el número de unidades de tiempo de cada una de las simulaciones que se van a realizar durante la simulación una vez iniciada la optimización. En este caso, las simulaciones se llevan a cabo utilizando el controlador.
- El número de épocas de entrenamiento del GP(`epochs_gp`): cada vez que se realice un entrenamiento introduciendo nuevos datos correspondientes a la última iteración, se ejecutará este número de épocas de entrenamiento del GP de forma interna.
- El número de épocas de optimización del controlador dentro de cada iteración de PILCO (`epochs_pilco`).

2.– Generación de datos iniciales: la función de este paso no consiste en entrenar con estos datos el GP, sino en obtener unos datos que tengan las mismas características que los datos que van a ser utilizados en PILCO a fin de inicializar las variables internas del GP.

3.– Creación de la sesión de TensorFlow e inicialización del GP.

4.– Ejecución del algoritmo PILCO mediante la función `optimize` de dicho módulo.

5.– Elaboración de gráficos y animaciones.

Tanto los gráficos como las animaciones han sido generadas utilizando la librería de Python *Matplotlib*. Para la creación de la animación, ha sido creado un módulo Animation cuya tarea consiste en recibir los datos base de esta animación, y tras generarla, guardarla en formato *gif* o mostrarla por pantalla según convenga.

5.2.3. Resultados

Tras algunas ejecuciones del test, se han escogido ciertos ejemplos que resultan interesantes para ver el funcionamiento del algoritmo y comprender su potencial. En los gráficos que se muestran a continuación se distingue la siguiente información:

- Posiciones obtenidas mediante la utilización de controles aleatorios durante la simulación inicial en color **gris**.
- Posiciones obtenidas mediante la utilización del controlador del sistema (la red neuronal) en color **negro**.
- Posición inicial de cada simulación en color **rojo**.

Se ha utilizado una línea a fin de visualizar más cómodamente la progresión de los movimientos obtenidos. En los casos en que se comunique un punto anterior con un punto de color rojo, significa que la simulación anterior ha finalizado y está comenzando una nueva simulación iniciada con una posición inicial aleatoria.

En todos los casos se puede observar que las posiciones generadas a partir de un controlador aleatorio son muy caóticas, esto es debido a la propia aleatoriedad del mismo. La realización de esta fase inicial es de bastante utilidad, ya que permite al GP aprender las dinámicas del sistema, es decir, los resultados esperados al aplicar un control específico sobre el carrito. Gracias esto, en las siguientes ejecuciones no es necesario hacer simulaciones muy largas, ya que el GP puede encargarse de modelar el sistema de forma artificial, obteniendo resultados en la función de coste que ayuden a entrenar el controlador.

En la ejecución mostrada en la figura B.6 se han utilizado unos parámetros medios o incluso algo bajos en algunos casos, que nos muestran unos resultados estándar sobre los que nos podemos apoyar a fin de establecer un punto de partida acerca del funcionamiento del sistema desarrollado. Podemos observar que una vez han sido generados los datos de la simulación inicial mediante el controlador aleatorio, los controles aplicados por el controlador inteligente tratan siempre se alcanzar la posición objetivo, acercándose cada vez más desde puntos más lejanos.

En la siguiente ejecución, mostrada en la figura B.7, han sido utilizados muy pocos valores en la simulación inicial: en lugar de 100 como en la ejecución anterior, esta vez la simulación ha sido realizada durante 10 unidades de tiempo. No obstante, se puede observar cómo el GP es capaz de aprender con suficiente precisión las dinámicas del entorno, como para entrenar el controlador de forma que consiga buenos resultados durante las simulaciones posteriores. Este ejemplo es especialmente importante, porque nos muestra el potencial que tiene el sistema PILCO, que con una cantidad muy baja de datos es capaz de obtener un controlador que se funcione de manera bastante positiva.

En la ejecución mostrada en la figura B.8 se ha hecho una simulación previa muy larga (1000 unidades de tiempo). Los movimientos realizados por el controlador, en este caso han sido más certeros en un número menor de simulaciones, pero no representan un gran cambio en comparación con las ejecuciones anteriores. Esto se debe a que, si bien es importante que el GP aprenda correctamente

las dinámicas del sistema, el controlador necesita un número mayor de iteraciones para mejorar sus movimientos.

Para la ejecución representada en la figura B.9 han sido utilizadas 1000 épocas tanto para el entrenamiento del GP, como para el número de épocas de optimización del controlador. Los resultados obtenidos son bastante buenos, debido a que con estos parámetros, el GP es capaz de aprender con bastante precisión las dinámicas del sistema, al mismo tiempo que se optimiza el controlador para aplicar movimientos mucho más certeros, que colocan el carrito en la posición objetivo con mucha más rapidez que en las ejecuciones anteriores.

5.2.4. Conclusiones

Tras los experimentos realizados hemos podido ver el comportamiento del sistema PILCO sobre el problema descrito en detalle, a continuación se procede a comentar algunas conclusiones alcanzadas.

La principal observación que llama la atención de las ejecuciones anteriores es la eficiencia de datos, ya que como hemos podido observar en el gráfico B.9, el sistema es capaz de aprender a controlar el carrito con un número de simulaciones y de unidades de tiempo muy bajos. Esta eficiencia se debe, como se ha comentado anteriormente, a la utilización del GP para modelar las dinámicas del sistema, ya que gracias a ello podemos hacer simulaciones artificiales de forma casi independiente al medio, de forma que el impacto en el entorno durante el aprendizaje del controlador es muy bajo.

De igual forma hemos visto que, aunque una cantidad superior de datos beneficia el aprendizaje del GP, el funcionamiento del mismo es suficientemente bueno con una cantidad reducida de datos como para dedicar los recursos computacionales a otras actividades que forman parte del proceso, como es el entrenamiento del controlador, sobre el que recae la mayor importancia una vez han sido aprendidas las dinámicas del entorno por el GP.

6

Conclusiones y Trabajo Futuro

6.0.1. Conclusiones

Los problemas de control forman parte del gran número de problemas que son resolubles utilizando algoritmos y técnicas de aprendizaje automático. La aproximación llevada a cabo durante este trabajo explora una alternativa a la metodología utilizada usualmente en este ámbito, aprovechando conocimientos más utilizados en otros problemas, pero con poco recorrido en este. La idea de simular el entorno utilizando un proceso estocástico para aprender sólo aquellos aspectos con los que interactúa el objeto a controlar es brillante, pues no es necesario simular el entorno completo para resolver el problema y sería una tarea demasiado costosa en términos generales.

Por tanto, los Procesos Gaussianos juegan un papel esencial en este trabajo, ya que su manera de funcionar, adaptándose a las observaciones conocidas y disminuyendo con cada nuevo conocimiento la incertidumbre, encaja perfectamente con el rol que toma en el sistema. Por otro lado, la utilización de una red neuronal en el rol de controlador significa un acercamiento a las técnicas utilizadas normalmente en este tipo de problemas, haciendo que el sistema completo cobre un sentido mucho mayor al utilizar esta y no otra alternativa por su similitud con un controlador real.

Durante este trabajo, se ha podido ver cómo el sistema PILCO, con algunas modificaciones realizadas, es capaz de resolver problemas de control con una eficiencia de datos y una velocidad bastante destacables. Esto es posible gracias al algoritmo ideado, con los distintos tipos de simulación, el aprendizaje del GP y el del propio controlador, apoyándose en el conocimiento recogido por el propio sistema. Aunque esta idea ha sido aplicada en problemas de una complejidad media-baja, su funcionamiento es el mismo en problemas de mayor complejidad, con lo que puede ser de gran utilidad para resolver problemas más complejos.

Durante la realización de este trabajo se han logrado los objetivos descritos al principio del documento, destacando la ampliación de mi conocimiento en este área así como en la utilización de las herramientas usadas durante todo el proceso.

6.0.2. Trabajo Futuro

En futuros proyectos, podría ser interesante aplicar este sistema a problemas de control de mayor envergadura, demostrando el potencial del mismo en ámbitos en que podría acabar aplicándose finalmente. Si se tuviera un problema de dimensiones mayores, teniendo en cuenta también la velocidad, la gravedad u otras variables físicas por ejemplo, podrían utilizarse varios GPs para modelar cada una de las dimensiones del sistema, asegurando una imitación del entorno bastante precisa. Al igual, sería interesante comparar las diferencias entre utilizar un único GP para modelar problemas de grandes dimensiones frente a utilizar un GP por cada dimensión.

La utilización de un controlador de mayor complejidad sobre un entorno, así como de un objeto a controlar que tenga una repercusión mayor sobre el mismo, significaría un aumento de la complejidad del conjunto. Este aumento de la complejidad del problema no influiría en el algoritmo principal, pero sí sería necesario gestionar de manera distinta el GP o GPs que intervinieran en el aprendizaje de las dinámicas del sistema.

Bibliografía

[1] C. E. R. Marc Peter Deisenroth, "Pilco: A model-based and data-efficient approach to policy search," 2011. Enlace de acceso.

[2] "Machine learning control," Enlace de acceso.

[3] S. L. B. y. B. R. N. Thomas Duriez, *Machine Learning Control - Taming Nonlinear Dynamics and Turbulence*. Springer International Publishing, 2017.

[4] Y. Shweta Bhatt, "5 things you need to know about reinforcement learning," Enlace de acceso.

[5] J. G. Schneider, "Exploiting model uncertainty estimates for safe dynamic control learning," *IEENN*, 1997. Enlace de acceso.

[6] G. P. I. G. R. B. B. Depraetere, M. Liu, "Comparison of model-free and model-based methods for time optimal hit control of a badminton robot," December 2014. Enlace de acceso.

[7] J. F. López, "Proceso estocástico," Enlace de acceso.

[8] J. F. López, "Simulación de montecarlo," Enlace de acceso.

[9] "Método de montecarlo," Enlace de acceso.

[10] Ángel Franco García, "Los métodos de montecarlo," Enlace de acceso.

[11] J. Schmidhuber, "Who invented backpropagation?," 2014. Enlace de acceso.

[12] "Rectifier (neural networks)," Enlace de acceso.

[13] J. Brownlee, "A gentle introduction to the rectified linear unit (relu)," 2019. Enlace de acceso.

[14] C. E. R. y Christopher K. I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[15] S. Roweis, "Gaussian identities," 1999. Enlace de acceso.

[16] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

[17] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[18] "Covariance functions," Enlace de acceso.

[19] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[20] G. N. y. J. P. Marc Peter Deisenroth, *A Survey on Policy Search for Robotics*. now: the essence of knowledge, 2011.

[21] "Automatic differentiation," Enlace de acceso.

[22] "tf.compat.v1.train.adamoptimizer," Enlace de acceso.

[23] "Module: tf.keras.optimizers," Enlace de acceso.

[24] J. Brownlee, "Gentle introduction to the adam optimization algorithm for deep learning," July 2017. Enlace de acceso.

[25] "Introduction to gradients and automatic differentiation," Enlace de acceso.

Acrónimos

ECM Error Cuadratico Medio.

GP Gaussian Process.

GPs Gaussian Processes.

Leaky ReLU Leaky Rectified Linear Unit.

PILCO Inferencia Probabilistica para Aprendizaje de Problemas de Control.

RBF Radial Basis Function.

ReLU Rectified Linear Unit.

SVMs Maquinas de vectores de soporte o Support Vector Machine.

Apéndices

A

ALGORITMOS

A.0.1. Algoritmo de PILCO

init: Obtención de parámetros del controlador generando muestras de $\theta \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
Aplicación de un control aleatorio y guardado de datos.
repeat
    Aprendizaje de las dinámicas del sistema utilizando todos los datos (los últimos generados y los de iteraciones anteriores) mediante el Proceso Gaussiano.
    Obtención de un controlador que maneje el objeto.
    repeat
        Evaluación de la calidad de los controles aplicados por el controlador.
        Mejora del controlador mediante descenso por gradiente.
        Actualización de parámetros θ.
    until convergencia; **return** θ^*
    Modificación del controlador π^* de la forma $\pi^* \leftarrow \pi(\theta^*)$.
    Aplicación de π^* al sistema mediante una simulación y guardado de los datos obtenidos.
until tareas aprendidas

Algoritmo A.1: Funcionamiento del algoritmo de aprendizaje definido en PILCO.

A.0.2. Algoritmo de Simulación de ControlProblem

input : total_time. Tiempo total de la simulación
outputs: moves, targets. Pares $(posicion, movimiento)$ y targets generados

1 x_t = `generate_position()`
2 moves = []
3 targets = []
4 **forall** _ ***in*** `range(` *total_time* `)` **do**
5 u_t = `controller.get_predictions(` x_t `)`
6 moves $\leftarrow [x_t, u_t]$
7 x_{t+1} = `move(` x_t, u_t `)`
8 $x_t = x_{t+1}$ o límite más cercano, en caso que el movimiento haya provocado una nueva posición fuera de los límites establecidos
9 targets $\leftarrow x_t$
10 **end**
11 **return** moves, targets

Algoritmo A.2: Algoritmo correspondiente a la simulación codificado en ControlProblem.py

B

Figuras

B.0.1. Red Neuronal Utilizada

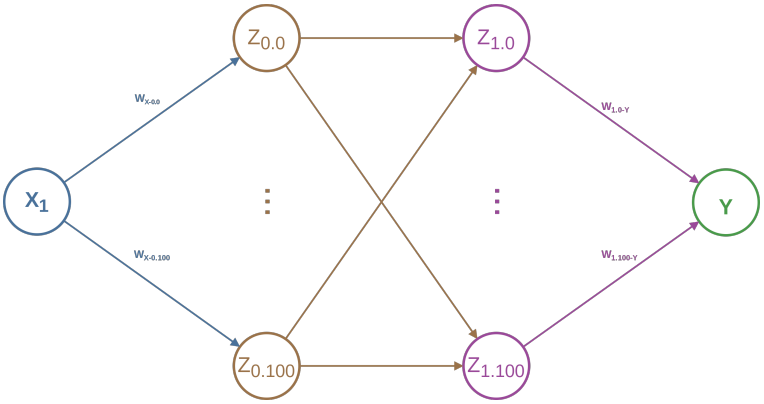


Figura B.1: Figura correspondiente a la red neuronal utilizada en las pruebas con una estructura de 1 neurona en la capa de entrada, dos capas ocultas de 100 neuronas cada una y una capa de salida de 1 neurona.

B.0.2. Gráficos Comparativos de los Parámetros de la Función de Covarianzas

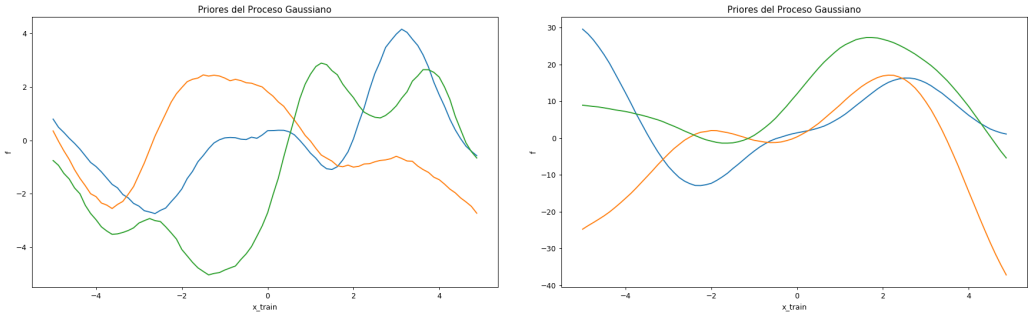


Figura B.2: Observaciones de 3 muestras del prior del GP utilizando $log\_lengthscales = 1$, $log\_sigma0 = -500{,}0$, $log\_sigma = 0{,}0$ y $log\_lengthscales = 5$, $log\_sigma0 = -500{,}0$, $log\_sigma = 0{,}0$.

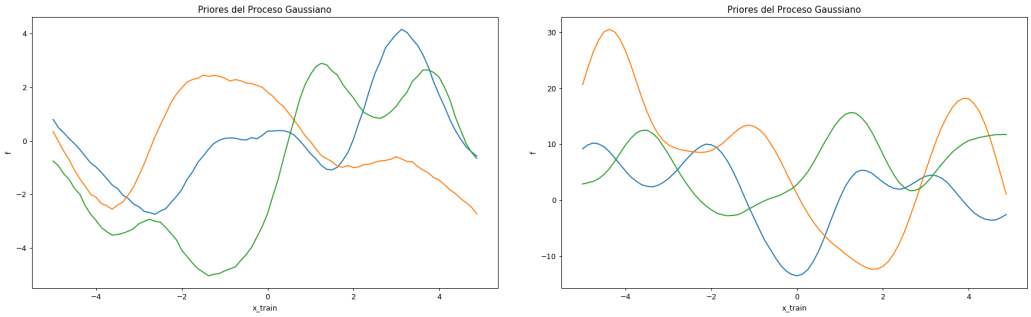


Figura B.3: Observaciones de 3 muestras del prior del GP utilizando $log\_lengthscales = 1$, $log\_sigma0 = -500{,}0$, $log\_sigma = 0{,}0$ y $log\_lengthscales = 1$, $log\_sigma0 = -500{,}0$, $log\_sigma = 6{,}0$.

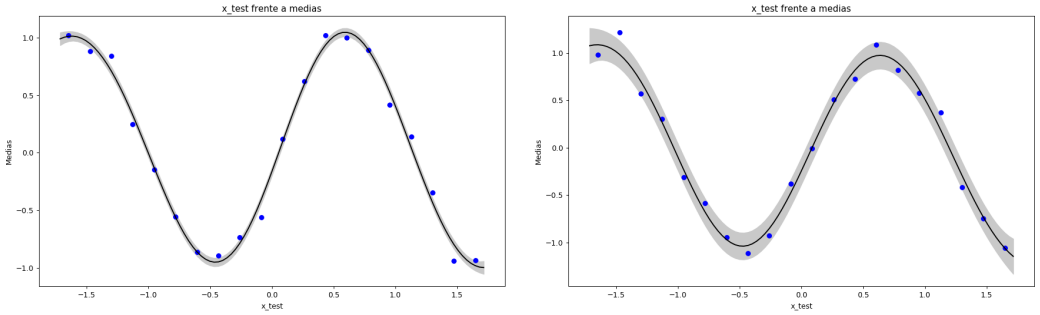


Figura B.4: Observaciones de la distribución predictiva del GP utilizando $log\_lengthscales = 1$, $log\_sigma0 = 0{,}0$, $log\_sigma = 0{,}0$ y $log\_lengthscales = 0$, $log\_sigma0 = 50{,}0$, $log\_sigma = 0{,}0$.

B.0.3. Gráfico de Tiempos de Ejecución PILCO

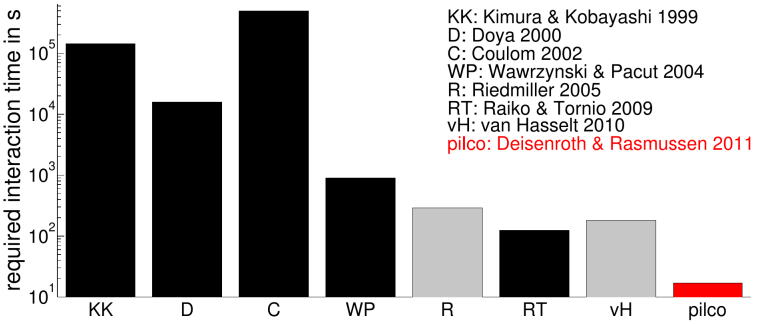


Figura B.5: Gráfico comparativo del tiempo necesario para resolver el problema *cart-pole* por algunos modelos de aprendizaje por refuerzo y PILCO.

B.0.4. Ejecuciones de PILCO

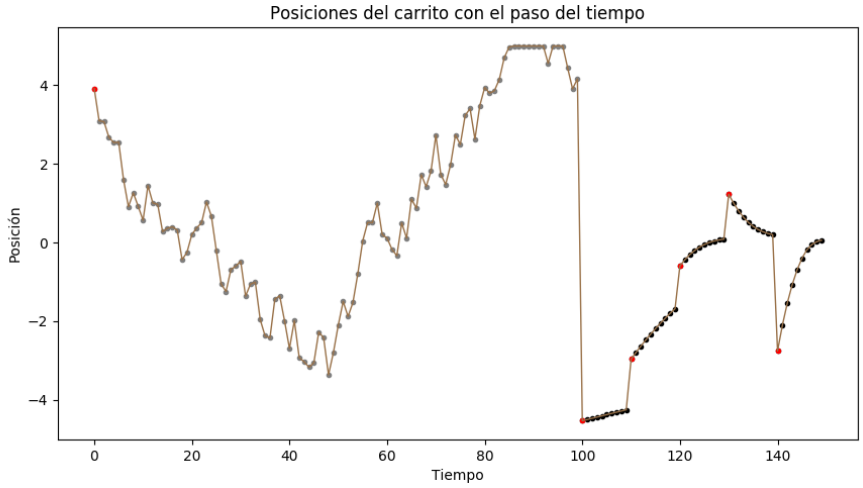


Figura B.6: Ejemplo de ejecución de PILCO con los parámetros: target_position = 0.0, total_time_ini = 100, total_time_iter = 10, epochs_gp = 100, epochs_pilco = 100.

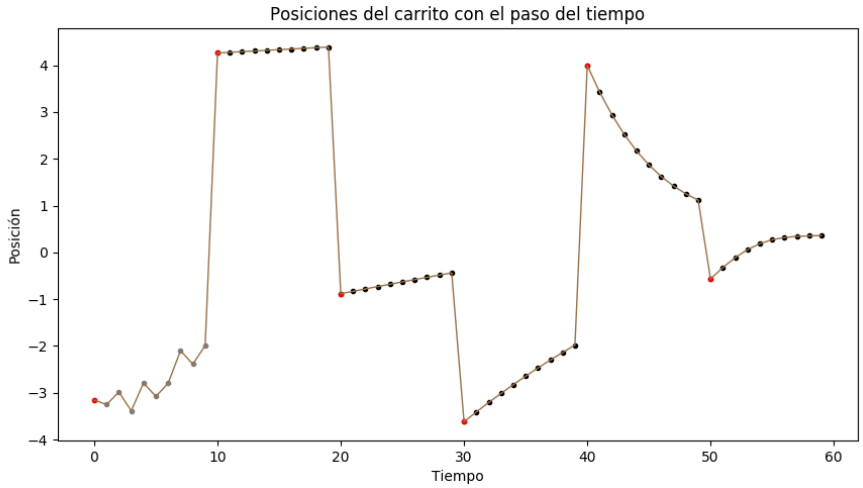


Figura B.7: Ejemplo de ejecución de PILCO con los parámetros: target_position = 0.0, total_time_ini = 10, total_time_iter = 10, epochs_gp = 100, epochs_pilco = 100.

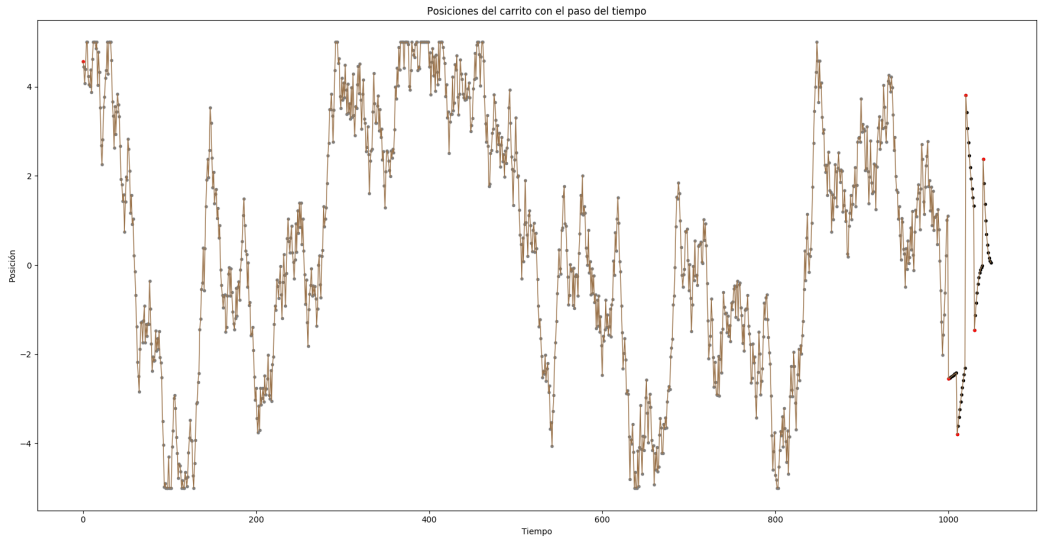


Figura B.8: Ejemplo de ejecución de PILCO con los parámetros: target_position = 0.0, total_time_ini = 1000, total_time_iter = 10, epochs_gp = 100, epochs_pilco = 100.

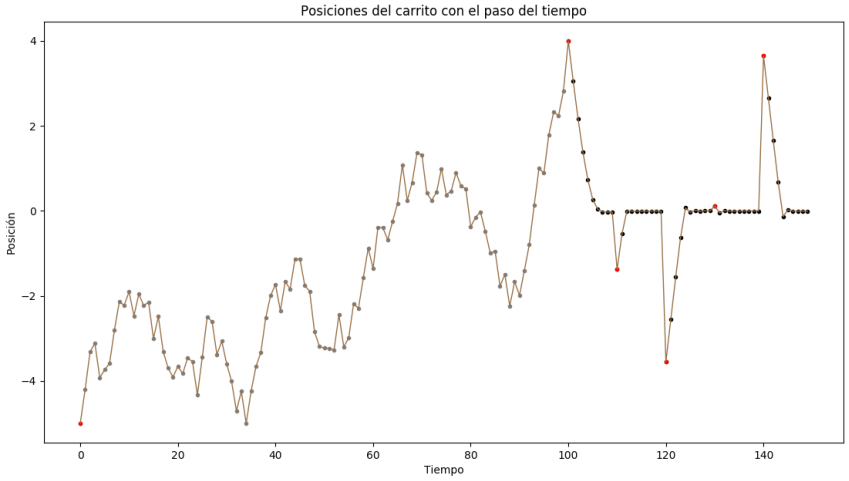


Figura B.9: Ejemplo de ejecución de PILCO con los parámetros: target_position = 0.0, total_time_ini = 100, total_time_iter = 10, epochs_gp = 1000, epochs_pilco = 1000.

C

Fragmentos de Código

C.0.1. Inicialización del GP en su Implementación

Código C.1: Este fragmento corresponde a la declaración de PlaceHolders de TensorFlow durante la inicialización del GP.

```
42        self.pl_inputs_train = tf.placeholder(tf.float32, shape=[None, dim_x], name="train_inputs")
43        self.pl_inputs_test = tf.placeholder(tf.float32, shape=[None, dim_x], name="test_inputs")
44        self.pl_outputs_train = tf.placeholder(tf.float32, shape=[None, 1], name="train_outputs")
```

Inicialización de la Red Neuronal en su Implementación

Código C.2: Este fragmento corresponde a la declaración de la estructura de red, inicializando las conexiones con valores aleatorios pequeños.

```
27        self.layers = [[tf.Variable(tf.multiply(x=tf.random.normal(shape=(m, n),
28                                                                   dtype=np.float32,
29                                                                   mean=0.0,
30                                                                   stddev=1.0),
31                                                y=1e-2)),
32                        tf.Variable(tf.multiply(x=tf.random.normal(shape=(n,),
33                                                                   dtype=np.float32,
34                                                                   mean=0.0,
35                                                                   stddev=1.0),
36                                                y=1e-2))]
37                       for m, n in zip(layer_sizes[:-1], layer_sizes[1:])]
```

Código C.3: Este fragmento corresponde a la declaración del PlaceHolder de los inputs de test de la red neuronal.

```
43        self.pl_inputs_test = tf.placeholder(tf.float32, shape=[None, layer_sizes[0]], name="test_inputs")
```

C.0.2. Implementación de la Optimización del Controlador

Código C.4: Este fragmento corresponde a la optimización del controlador y el GP en Pilco.py.

```
0  def optimize(self, total_time_ini, total_time_iter):
1      # Generacion de datos de train y Entrenamiento del GP
2      inputs_tt, outputs_tt = self.control_problem.simulate(total_time_ini, random_control=True)
3
4      # Entrenamiento del GP
5      self.fit_GP(inputs_tt, outputs_tt)
6
7      # Calculo de J(theta) y obtencion del coste C
8      C = self.get_cost(total_time_iter)
9
10     # Creacion del optimizador
11     train_step = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(loss=C,
           var_list=self.controller.layers)
12
13     # Inicializacion de variables de Adam
14     self.session.run(tf.global_variables_initializer())
15
16     # Reentrenamiento del GP porque al inicializar las variables se han reseteado las variables
17     self.fit_GP(inputs_tt, outputs_tt)
18
19     for _ in np.arange(self.opt_rounds):
20
21         cost = 0.0
22
23         for epoch in np.arange(self.epochs):
24
25             cost += self.session.run(C, feed_dict={self.GP.pl_inputs_train: inputs_tt, \
26                                                    self.GP.pl_outputs_train: outputs_tt})
27             self.session.run(train_step, feed_dict={self.GP.pl_inputs_train: inputs_tt, \
28                                                    self.GP.pl_outputs_train: outputs_tt})
29
30         # Generacion de datos de train y Entrenamiento del GP
31         inputs, targets = self.control_problem.simulate(total_time_iter, random_control=False)
32         inputs_tt = np.concatenate((inputs_tt, inputs))
33         outputs_tt = np.concatenate((outputs_tt, targets))
34
35         self.fit_GP(inputs_tt, outputs_tt)
36
37     return inputs_tt, outputs_tt
```


UAM
UNIVERSIDAD AUTONOMA
DE MADRID